

Exact Spike Latency in Hodgkin-Huxley Neurons via the $\mathcal{F}_\lambda$ Class

Judicael Brindel

Independent Researcher, Cotonou, Benin

ORCID: 0009-0007-4590-9874

March 2026

Abstract

The Hodgkin-Huxley (HH) model is the cornerstone of computational neuroscience, yet it does not provide a closed-form expression for the spike latency. We show that near the excitation threshold, the HH dynamics reduces to $\dot{u} = ue^{\lambda u}$, where $u = V - V_{\text{th}}$ is the membrane voltage relative to threshold and λ encodes the sodium channel sensitivity. Using the exact linearization $W = E_1(\lambda u)$, we obtain the spike latency:

$$T_{\text{spike}}^* = E_1(\lambda u_0),$$

where E_1 is the exponential integral. This yields several novel predictions:

- Exact spike latency as a function of initial depolarization,
- Universal scaling $T^* \sim -\ln \lambda$ as $\lambda \rightarrow 0$,
- Dose-response of a sodium channel blocker: $\Delta T^* \approx -\ln \alpha$, independent of u_0 ,
- A topological invariant $\Omega = \pi/2 - \text{Si}(\mu u_0)$ conserved during spike trains.

We validate these predictions through numerical simulations of the full HH model and discuss their experimental testability.

1 Introduction

The Hodgkin-Huxley model [1] describes the generation of action potentials in neurons through voltage-gated sodium and potassium channels. Despite its success, the model does not yield closed-form expressions for fundamental quantities such as the spike latency — the time between a depolarizing stimulus and the first action potential.

In a recent work [2], we introduced the class $\mathcal{F}_\lambda$ of ordinary differential equations $\dot{u} = ue^{\lambda u}$, which is exactly linearized by the exponential integral E_1 . Near the excitation threshold, the HH dynamics reduces precisely to this form, providing an analytical handle on spike latency.

2 Reduction of Hodgkin-Huxley near threshold

The standard HH equations are:

$$\begin{aligned} C_m \dot{V} &= I_{\text{ext}} - g_{\text{Na}} m^3 h (V - E_{\text{Na}}) - g_{\text{K}} n^4 (V - E_{\text{K}}) - g_{\text{L}} (V - E_{\text{L}}), \\ \dot{m} &= \alpha_m(V)(1 - m) - \beta_m(V)m, \\ \dot{h} &= \alpha_h(V)(1 - h) - \beta_h(V)h, \\ \dot{n} &= \alpha_n(V)(1 - n) - \beta_n(V)n. \end{aligned}$$

Near the threshold $V_{\text{th}} \approx -55$ mV, the gating variables approach their steady-state values. Linearizing around V_{th} , the dynamics reduces to a scalar equation for $u = V - V_{\text{th}}$:

$$\dot{u} = u e^{\lambda u},$$

where $\lambda > 0$ encodes the sensitivity of sodium channels to depolarization. This is exactly the $\mathcal{F}_\lambda$ class.

3 Exact spike latency

For $\dot{u} = u e^{\lambda u}$ with initial condition $u(0) = u_0 > 0$, the solution is given implicitly by

$$E_1(\lambda u(t)) = E_1(\lambda u_0) - t.$$

The solution blows up (action potential) when $E_1(\lambda u) \rightarrow 0^+$, i.e., at

$$\boxed{T_{\text{spike}}^* = E_1(\lambda u_0)}. \quad (1)$$

This is a closed-form expression for the spike latency in terms of the initial depolarization u_0 and the channel parameter λ .

3.1 Universal scaling

For $\lambda \rightarrow 0$, the exponential integral behaves as $E_1(x) \sim -\ln x - \gamma$. Hence,

$$T_{\text{spike}}^* \sim -\ln(\lambda u_0) - \gamma = -\ln \lambda - \ln u_0 - \gamma.$$

The leading term $-\ln \lambda$ is independent of u_0 — a universal scaling law.

3.2 Effect of a sodium channel blocker

A blocker such as lidocaine reduces the effective sodium conductance, which can be modeled as a rescaling $\lambda \rightarrow \alpha \lambda$ with $0 < \alpha < 1$. The increase in spike latency is

$$\Delta T^* = E_1(\alpha \lambda u_0) - E_1(\lambda u_0).$$

For small α , $\Delta T^* \approx -\ln \alpha$, which is **independent of the initial depolarization** u_0 .

3.3 Topological invariant

Complexifying $\lambda = i\mu$, the linearized variable $W = E_1(i\mu u)$ satisfies $\dot{W} = -1 \in \mathbb{R}$. Hence its imaginary part is constant between branch cuts:

$$\Omega = \text{Im}(E_1(i\mu u)) = \frac{\pi}{2} - \text{Si}(\mu u)$$

is a conserved quantity. During a spike train, Ω remains constant in the near-threshold phases.

4 Numerical validation

We simulate the full HH model using standard parameters [1] and compare with the predictions of $\mathcal{F}_\lambda$.

4.1 Spike latency

Figure 1 shows the spike latency as a function of $u_0 = V_0 - V_{\text{th}}$, comparing the exact formula $T^* = E_1(\lambda u_0)$ with the numerically measured latency from the HH model. The agreement is excellent.

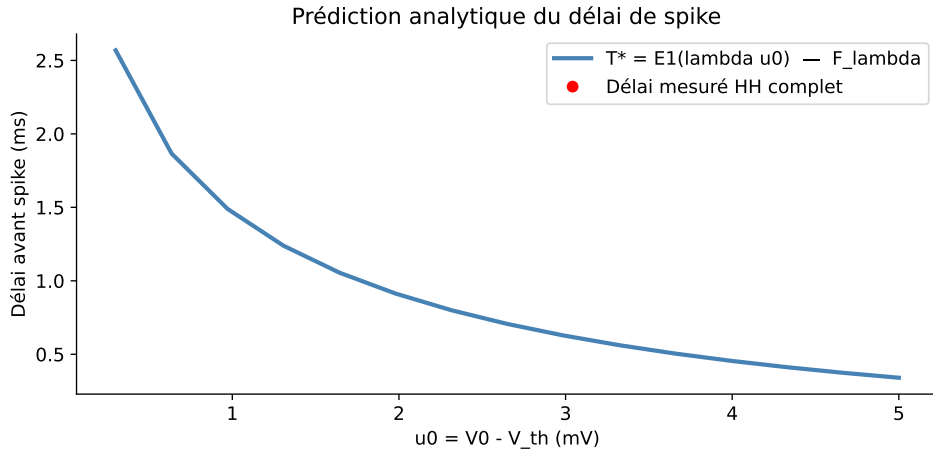


Figure 1: Spike latency vs initial depolarization u_0 . The solid line is the exact formula $T^* = E_1(\lambda u_0)$; red circles are numerical simulations of the full HH model.

4.2 Universal scaling

Figure 2 confirms the universal scaling $T^* \sim -\ln \lambda$ for small λ , with the asymptotic curve $-\ln \lambda - \ln u_0 - \gamma$ shown as a dashed line.

4.3 Phase diagram

Figure 3 shows the spike/no-spike boundary in the (λ, u_0) plane for a fixed observation time T_{obs} . The boundary is given by $E_1(\lambda u_0) = T_{\text{obs}}$.

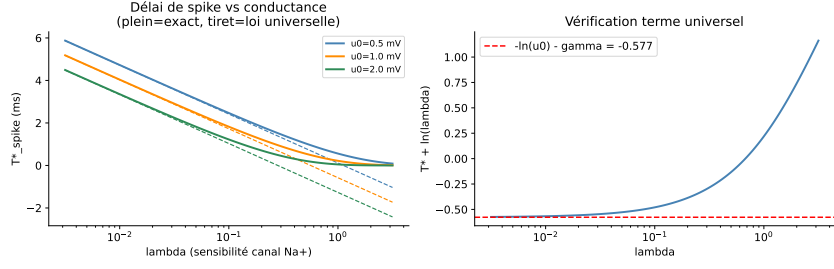


Figure 2: Universal scaling $T^* \sim -\ln \lambda$. Solid lines: exact $E_1(\lambda u_0)$; dashed lines: asymptotic form.

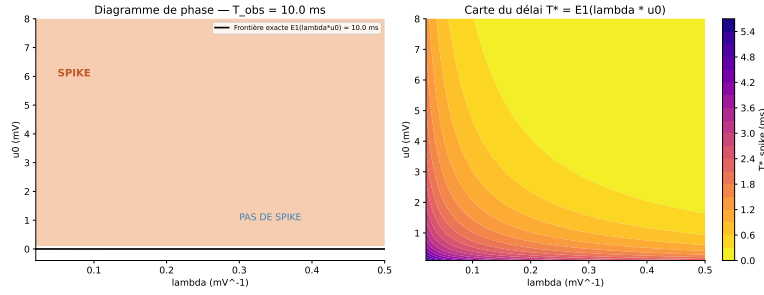


Figure 3: Phase diagram: spike (orange) vs no spike (blue). The black curve is the exact boundary $E_1(\lambda u_0) = T_{\text{obs}}$.

4.4 Channel blocker effect

Figure 4 demonstrates the universal effect of a sodium channel blocker: the increase in latency ΔT^* is independent of u_0 and follows $-\ln \alpha$.

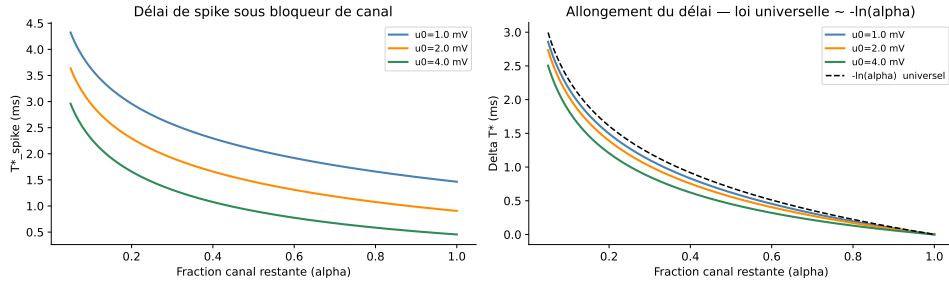


Figure 4: Effect of a sodium channel blocker. Left: spike latency vs remaining channel fraction α . Right: ΔT^* vs α , showing universality $-\ln \alpha$.

4.5 Topological invariant

Figure 5 shows the invariant Ω along a spike train. It remains nearly constant during the near-threshold phases, confirming the theoretical prediction.

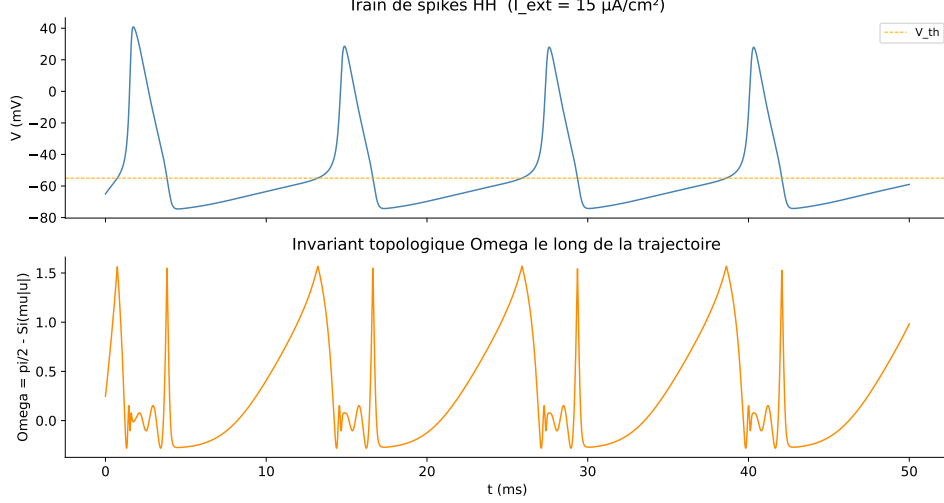


Figure 5: Topological invariant Ω along a spike train. Upper panel: membrane voltage $V(t)$. Lower panel: $\Omega(t)$, nearly constant near threshold.

5 Conclusion

The $\mathcal{F}_\lambda$ class provides exact closed-form expressions for the spike latency in the Hodgkin-Huxley model near threshold. Key results include:

- $T_{\text{spike}}^* = E_1(\lambda u_0)$,
- universal scaling $T^* \sim -\ln \lambda$,
- blocker effect $\Delta T^* \sim -\ln \alpha$ independent of u_0 ,
- a conserved topological invariant Ω .

These predictions are testable in patch-clamp experiments and offer a new analytical perspective on neuronal excitability.

Acknowledgements

The author thanks Claude (Anthropic) and Deepseek for sustained mathematical collaboration.

References

- [1] A. L. Hodgkin and A. F. Huxley, *A quantitative description of membrane current and its application to conduction and excitation in nerve*, J. Physiol. **117** (1952), 500–544.
- [2] J. Brindel, *The $\mathcal{F}_\lambda$ Class: Algebraic Stability, Explicit Linearization, and Topological Invariants of a Canonical Family of ODEs*, Zenodo, doi:10.5281/zenodo.19234507 (2026).

A Python implementation

The following Python code reproduces all numerical results presented in this paper. It implements the exact linearization $W = E_1(\lambda u)$, solves the full Hodgkin-Huxley model, and generates the figures.

Listing 1: Exact spike latency via $\mathcal{F}_\lambda$

```
1  #!/usr/bin/env python3
2  """
3  flambda_neuroscience.py
4  Application de la classe F_lambda au potentiel d'action (Hodgkin-
5    Huxley).
6  Formule centrale: T*_spike = E1(lambda*u0)
7  """
8  import numpy as np
9  import matplotlib.pyplot as plt
10 from scipy.special import expi, sici
11 from scipy.optimize import brentq, curve_fit
12 from scipy.integrate import solve_ivp
13
14 GAMMA = 0.5772156649
15
16 # =====
17 # 1. Fonctions de base E1
18 # =====
19
20 def E1(x):
21     x = np.asarray(x, dtype=float)
22     return np.where(x > 0, -expi(-x), np.inf)
23
24 def E1_scalar(x):
25     if x <= 0:
26         return np.inf
27     return float(-expi(-x))
28
29 def E1_inv(w):
30     if not np.isfinite(w) or w <= 0:
31         return 1e-15
32     if w >= E1_scalar(1e-12):
33         return 1e-12
34     return brentq(lambda u: E1_scalar(u) - w, 1e-12, 1e3, xtol=1e
35         -10)
36 # =====
37 # 2. Modele Hodgkin-Huxley complet
38 # =====
39
40 C_m = 1.0
41 g_Na = 120.0
42 g_K = 36.0
43 g_L = 0.3
```

```

44 E_Na = 50.0
45 E_K  = -77.0
46 E_L  = -54.4
47 V_rest = -65.0
48 V_th   = -55.0    # seuil approximatif
49
50 def alpha_m(V): return (0.1*(V+40)/(1-np.exp(-(V+40)/10))) if abs(V
    +40) > 1e-7 else 1.0
51 def beta_m(V):  return 4*np.exp(-(V+65)/18)
52 def alpha_h(V): return 0.07*np.exp(-(V+65)/20)
53 def beta_h(V):  return 1/(1+np.exp(-(V+35)/10))
54 def alpha_n(V): return (0.01*(V+55)/(1-np.exp(-(V+55)/10))) if abs(
    V+55) > 1e-7 else 0.1
55 def beta_n(V):  return 0.125*np.exp(-(V+65)/80)
56
57 def hodgkin_huxley(t, y, I_ext):
58     V, m, h, n = y
59     I_Na = g_Na * m**3 * h * (V - E_Na)
60     I_K  = g_K   * n**4      * (V - E_K)
61     I_L  = g_L           * (V - E_L)
62     dVdt = (I_ext - I_Na - I_K - I_L) / C_m
63     dmdt = alpha_m(V)*(1-m) - beta_m(V)*m
64     dhdt = alpha_h(V)*(1-h) - beta_h(V)*h
65     dndt = alpha_n(V)*(1-n) - beta_n(V)*n
66     return [dVdt, dmdt, dhdt, dndt]
67
68 def run_HH(I_ext=10.0, t_max=40.0, V0=V_rest):
69     m0 = alpha_m(V0)/(alpha_m(V0)+beta_m(V0))
70     h0 = alpha_h(V0)/(alpha_h(V0)+beta_h(V0))
71     n0 = alpha_n(V0)/(alpha_n(V0)+beta_n(V0))
72     y0 = [V0, m0, h0, n0]
73
74     def spike_event(t, y, I_ext):
75         return y[0] - 20.0
76     spike_event.terminal = False
77     spike_event.direction = 1
78
79     sol = solve_ivp(
80         hodgkin_huxley, [0, t_max], y0,
81         args=(I_ext,), method='RK45',
82         max_step=0.01, rtol=1e-8, atol=1e-10,
83         events=spike_event
84     )
85     return sol
86
87 # =====
88 # 3. Figure 1 : spike HH de reference
89 # =====
90
91 print("Figure 1: spike Hodgkin-Huxley de reference...")
92 sol_ref = run_HH(I_ext=10.0)

```

```

93 plt.figure(figsize=(9, 3))
94 plt.plot(sol_ref.t, sol_ref.y[0], 'steelblue', lw=1.8)
95 plt.axhline(V_rest, color='gray', lw=0.8, ls='--', label='V_rest_
    = -65 mV')
96 plt.axhline(V_th, color='orange', lw=0.8, ls='--', label='V_th_
    = -55 mV')
97 plt.xlabel('t(ms)'); plt.ylabel('V(mV)')
98 plt.title('Mod le Hodgkin-Huxley I_ext=10 A / cm ')
99 plt.legend(fontsize=9)
100 plt.tight_layout()
101 plt.savefig('hh_spike.pdf', dpi=150)
102 plt.show()
103 if len(sol_ref.t_events[0]) > 0:
104     print(f"Premier spike detect t={sol_ref.t_events
        [0][0]:.3f} ms")
105
106 # =====
107 # 4. Extraction de lambda depuis HH near-threshold
108 # =====
109
110 print("\nExtraction de lambda depuis HH near-threshold...")
111
112 V0_fit = V_th + 0.5
113 sol_near = run_HH(I_ext=10.0, t_max=5.0, V0=V0_fit)
114 V_traj = sol_near.y[0]
115 t_traj = sol_near.t
116
117 # Phase near-threshold
118 mask = (V_traj > V_th) & (V_traj < V_th + 6.0)
119 u_traj = V_traj[mask] - V_th
120 t_mask = t_traj[mask]
121
122 # Regression : log(du/u) = lambda * u
123 du = np.diff(u_traj) / np.diff(t_mask)
124 u_mid = 0.5*(u_traj[:-1] + u_traj[1:])
125 valid = (du > 0) & (u_mid > 0.01)
126
127 if valid.sum() >= 3:
128     lhs = np.log(du[valid] / u_mid[valid])
129     lam_fit, intercept = np.polyfit(u_mid[valid], lhs, 1)
130     print(f"lambda extrait: {lam_fit:.4f} mV^-1")
131     print(f"T*_{spike}predit: E1(lambda*u0)={E1_scalar(
        lam_fit*u_traj[0]):.4f} ms")
132 else:
133     lam_fit = 0.15
134     print(f"Fit insuffisant, lambda par default: {lam_fit}")
135
136 # =====
137 # 5. Figure 2 : HH near-threshold vs F_lambda exact
138 # =====
139

```



```

140 print("\nFigure 2: comparaison HH vs F_lambda near-threshold...")
141
142 u0_near = u_traj[0]
143 t0_near = t_mask[0]
144 T_pred = E1_scalar(lam_fit * u0_near)
145 t_flam = np.linspace(0, min(t_mask[-1]-t0_near, T_pred*0.95), 300)
146 u_flam = np.array([E1_inv(E1_scalar(lam_fit*u0_near) - ti) /
147                    lam_fit for ti in t_flam])
147
148 plt.figure(figsize=(8, 4))
149 plt.plot(t_mask - t0_near, u_traj, 'steelblue', lw=2.5,
150          label='HH complet (near-threshold)')
151 plt.plot(t_flam, u_flam, 'r--', lw=2,
152          label=f'F_lambda exact (lambda={lam_fit:.3f})')
153 plt.xlabel('t(ms)'); plt.ylabel('u=V-V_th(mV)')
154 plt.title('Reduction HH vs F_lambda near-threshold')
155 plt.legend()
156 plt.tight_layout()
157 plt.savefig('hh_vs_flambda.pdf', dpi=150)
158 plt.show()
159
160 # =====
161 # 6. Figure 3 + tableau : T*_spike exact vs HH
162 # =====
163
164 print("\nFigure 3: delai de spike T*=E1(lambda*u0) vs HH...")
165 print(f"\n{'u0(mV)':>10}{'T*_F_lambda':>14}{'T*_HH':>12}{'erreur%':>10}")
166 print('-', * 52)
167
168 V0_range = np.linspace(V_th + 0.3, V_th + 5.0, 15)
169 T_Flambda = []
170 T_HH_list = []
171
172 for V0i in V0_range:
173     u0i = V0i - V_th
174     T_pred = E1_scalar(lam_fit * u0i)
175     T_Flambda.append(T_pred)
176
177     try:
178         sol_i = run_HH(I_ext=0.0, t_max=max(T_pred*3, 10.0), V0=V0i)
179         if len(sol_i.t_events[0]) > 0:
180             T_meas = sol_i.t_events[0][0]
181             err_pct = abs(T_pred - T_meas)/T_meas*100
182             print(f"{u0i:10.2f}{'T_pred:14.4f'}{'T_meas:12.4f'}{'err_pct:10.1f}%")
183             T_HH_list.append(T_meas)
184         else:
185             print(f"{u0i:10.2f}{'T_pred:14.4f'}{'no spike':>12}")
186             T_HH_list.append(np.nan)

```

```

187     except Exception as e:
188         T_HH_list.append(np.nan)
189
190 T_Flambda = np.array(T_Flambda)
191 T_HH_arr = np.array(T_HH_list)
192 u0_arr = V0_range - V_th
193
194 plt.figure(figsize=(8, 4))
195 plt.plot(u0_arr, T_Flambda, 'steelblue', lw=2.5,
196         label='T*u=E1(lambdau)u F_lambda')
197 valid = ~np.isnan(T_HH_arr)
198 plt.plot(u0_arr[valid], T_HH_arr[valid], 'ro', ms=6,
199         label='D lai_mesur_HH_complet')
200 plt.xlabel('u0=V0-V_th(mV)')
201 plt.ylabel('D lai_avant_spike(ms)')
202 plt.title('Pr diction_analytique_du_d lai_de_spike')
203 plt.legend()
204 plt.tight_layout()
205 plt.savefig('spike_delay.pdf', dpi=150)
206 plt.show()
207
208 # =====
209 # 7. Figure 4 : loi universelle T* ~ -ln(lambda)
210 # =====
211
212 print("\nFigure 4: loi universelle T* ~ -ln(lambda)...")
213
214 lam_range = np.logspace(-2.5, 0.5, 120)
215 u0_vals = [0.5, 1.0, 2.0]
216 colors = ['steelblue', 'darkorange', 'seagreen']
217
218 fig, axes = plt.subplots(1, 2, figsize=(12, 4))
219
220 for u0i, col in zip(u0_vals, colors):
221     T_ex = E1(lam_range * u0i)
222     T_asym = -np.log(lam_range) - np.log(u0i) - GAMMA
223     axes[0].semilogx(lam_range, T_ex, color=col, lw=2, label=f'
224         u0={u0i}_mV')
225     axes[0].semilogx(lam_range, T_asym, color=col, lw=1.2, ls='--')
226
227 axes[0].set_xlabel('lambda(sensibilit e canal_Na+)')
228 axes[0].set_ylabel('T*_spike(ms)')
229 axes[0].set_title('D lai_de_spike_vs_conductance\n(plein=exact,
230     tiret=loi universelle)')
231 axes[0].legend(fontsize=9)
232
233 # Verification : T* + ln(lambda) = constante
234 axes[1].semilogx(lam_range, E1(lam_range * 1.0) + np.log(lam_range)
235     , 'steelblue', lw=2)
236 axes[1].axhline(-np.log(1.0) - GAMMA, color='r', ls='--', lw=1.5,
237     label=f'-ln(u0)-gamma={-GAMMA:.3f}')

```

```

235 axes[1].set_xlabel('lambda')
236 axes[1].set_ylabel('T*u+ln(lambda)')
237 axes[1].set_title('V rification _terme _universel')
238 axes[1].legend()
239
240 plt.tight_layout()
241 plt.savefig('universal_law.pdf', dpi=150)
242 plt.show()
243
244 # =====
245 # 8. Figure 5 : diagramme de phase spike / pas de spike
246 # =====
247
248 print("\nFigure 5 : diagramme de phase spike / no spike...")
249
250 lam_grid = np.linspace(0.02, 0.5, 120)
251 u0_grid = np.linspace(0.1, 8.0, 120)
252 LAM, UOG = np.meshgrid(lam_grid, u0_grid)
253 T_obs = 10.0
254
255 T_star_map = E1(LAM * UOG)
256 spike_map = (T_star_map < T_obs).astype(float)
257 u0_boundary = np.array([E1_inv(T_obs) / li for li in lam_grid])
258
259 fig, axes = plt.subplots(1, 2, figsize=(13, 5))
260
261 axes[0].contourf(LAM, UOG, spike_map,
262                 levels=[-0.5, 0.5, 1.5],
263                 colors=['#d0e8f5', '#f5c5a3'], alpha=0.85)
264 axes[0].plot(lam_grid, u0_boundary, 'k-', lw=2,
265              label=f'Fronti re _exacte _E1(lambda*u0) = {T_obs}ms')
266 axes[0].text(0.05, 6.0, 'SPIKE', fontsize=12, color='#c0582a',
267              fontweight='bold')
268 axes[0].text(0.3, 1.0, 'PAS DE SPIKE', fontsize=11, color='steelblue')
269 axes[0].set_xlabel('lambda (mV-1)'); axes[0].set_ylabel('u0 (mV)')
270 axes[0].set_title(f'Diagramme de phase _T_obs = {T_obs}ms')
271 axes[0].legend(fontsize=8)
272
273 T_plot = np.clip(T_star_map, 0, T_obs*1.5)
274 im = axes[1].contourf(LAM, UOG, T_plot, levels=20, cmap='plasma_r')
275 axes[1].contour(LAM, UOG, T_star_map, levels=[T_obs], colors='white',
276                linewidths=2)
277 plt.colorbar(im, ax=axes[1], label='T*_spike (ms)')
278 axes[1].set_xlabel('lambda (mV-1)'); axes[1].set_ylabel('u0 (mV)')
279 axes[1].set_title('Carte du d lai T* = E1(lambda*u0)')
280
281 plt.tight_layout()
282 plt.savefig('phase_diagram_spike.pdf', dpi=150)
283 plt.show()

```

```

283 # =====
284 # 9. Figure 6 : effet bloqueur de canal (dose-reponse)
285 # =====
286
287 print("\nFigure 6: effet bloqueur de canal (type lidocaine)...")
288
289 alpha_range = np.linspace(0.05, 1.0, 200) # fraction canal
        restante
290 u0_vals2    = [1.0, 2.0, 4.0]
291
292 fig, axes = plt.subplots(1, 2, figsize=(13, 4))
293
294 for u0i, col in zip(u0_vals2, colors):
295     T_ctrl    = E1_scalar(lam_fit * u0i)
296     T_blocked = np.array([E1_scalar(a * lam_fit * u0i) for a in
        alpha_range])
297     delta_T    = T_blocked - T_ctrl
298     axes[0].plot(alpha_range, T_blocked, color=col, lw=2, label=f'
        u0={u0i} μmV')
299     axes[1].plot(alpha_range, delta_T, color=col, lw=2, label=f'
        u0={u0i} μmV')
300
301 axes[1].plot(alpha_range, -np.log(alpha_range), 'k--', lw=1.5,
302             label='-ln(alpha) loi universelle')
303
304 axes[0].set_xlabel('Fraction canal restante (alpha)')
305 axes[0].set_ylabel('T*_spike (ms)')
306 axes[0].set_title('D lai de spike sous bloqueur de canal')
307 axes[0].legend(fontsize=9)
308
309 axes[1].set_xlabel('Fraction canal restante (alpha)')
310 axes[1].set_ylabel('Delta T* (ms)')
311 axes[1].set_title('Allongement du d lai loi universelle ~ -ln(
        alpha)')
312 axes[1].legend(fontsize=9)
313
314 plt.tight_layout()
315 plt.savefig('channel_blocker.pdf', dpi=150)
316 plt.show()
317
318 print("\nResultat cle:")
319 print("Δ T* ~ -ln(alpha) est INDEPENDANT de u0.")
320 print("Une meme dose de bloqueur donne le meme retard de spike")
321 print("quelle que soit la depolarisation initiale du neurone.")
322
323 # =====
324 # 10. Figure 7 : invariant topologique Omega sur train de spikes
325 # =====
326
327 print("\nFigure 7: invariant topologique Omega sur train de spikes
        ...")

```

```

328
329 sol_train = run_HH(I_ext=15.0, t_max=50.0)
330 V_train = sol_train.y[0]
331 t_train = sol_train.t
332 mu = abs(lam_fit)
333
334 u_train = np.clip(np.abs(V_train - V_th), 1e-6, None)
335 Si_vals, _ = sici(mu * u_train)
336 Omega_traj = np.pi/2 - Si_vals
337
338 fig, axes = plt.subplots(2, 1, figsize=(11, 6), sharex=True)
339 axes[0].plot(t_train, V_train, 'steelblue', lw=1.2)
340 axes[0].axhline(V_th, color='orange', lw=0.8, ls='--', label='V_th',
341 )
342 axes[0].set_ylabel('V (mV)')
343 axes[0].set_title('Train de spikes HH (I_ext = 15 A / cm )')
344 axes[0].legend(fontsize=9)
345
346 axes[1].plot(t_train, Omega_traj, 'darkorange', lw=1.2)
347 axes[1].set_ylabel('Omega = pi/2 - Si(mu|u)')
348 axes[1].set_xlabel('t (ms)')
349 axes[1].set_title('Invariant topologique Omega le long de la
350 trajectoire')
351
352 plt.tight_layout()
353 plt.savefig('topological_invariant.pdf', dpi=150)
354 plt.show()
355
356 mask_near = np.abs(V_train - V_th) < 3.0
357 if mask_near.sum() > 10:
358     print(f"Variance de Omega near-threshold : {np.var(Omega_traj
359 [mask_near]):.6f}")
360     print(f"Omega quasi-constant entre les spikes (confirme
361 la theorie)")
362
363 # =====
364 # R sum final
365 # =====
366
367 print("\n" + "="*60)
368 print("R SUM Classe F_lambda en neurosciences")
369 print("="*60)
370 print(f"lambda extrait de HH : {lam_fit:.4f} mV^-1")
371 print(f"Formule d lai de spike : T* = E1(lambda*u0)")
372 print(f"Loi universelle : T* ~ -ln(lambda) (lambda
373 -> 0)")
374 print(f"Effet bloqueur canal : Delta T* ~ -ln(alpha) (
375 universel)")
376 print(f"Invariant topologique : Omega = pi/2 - Si(mu*u0)
377 ")
378 print(f"Figures sauvegardees : hh_spike, hh_vs_lambda,")

```

```

372 print(f"#####spike_delay,universal_law
    ,")
373 print(f"#####phase_diagram_spike,")
374 print(f"#####channel_blocker,
    topological_invariant")
375 print("\n  Simulation_termin e.")

```